Parveen Khurana

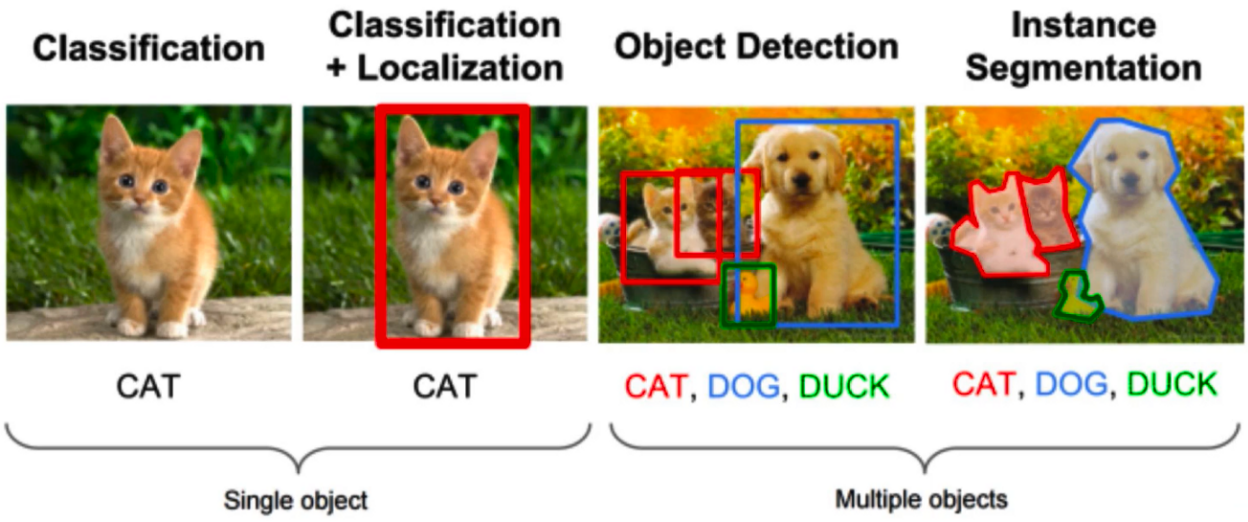
124 Followers About Following

CNN Architectures

P Parveen Khurana Feb 16, 2020 · 8 min read

This article covers the content discussed in the CNN Architectures module of the Deep Learning course and all the images are taken from the same module.

Kind of tasks CNNs are used for: Typically CNNs are used for various image tasks and most popular category of tasks are depicted below:



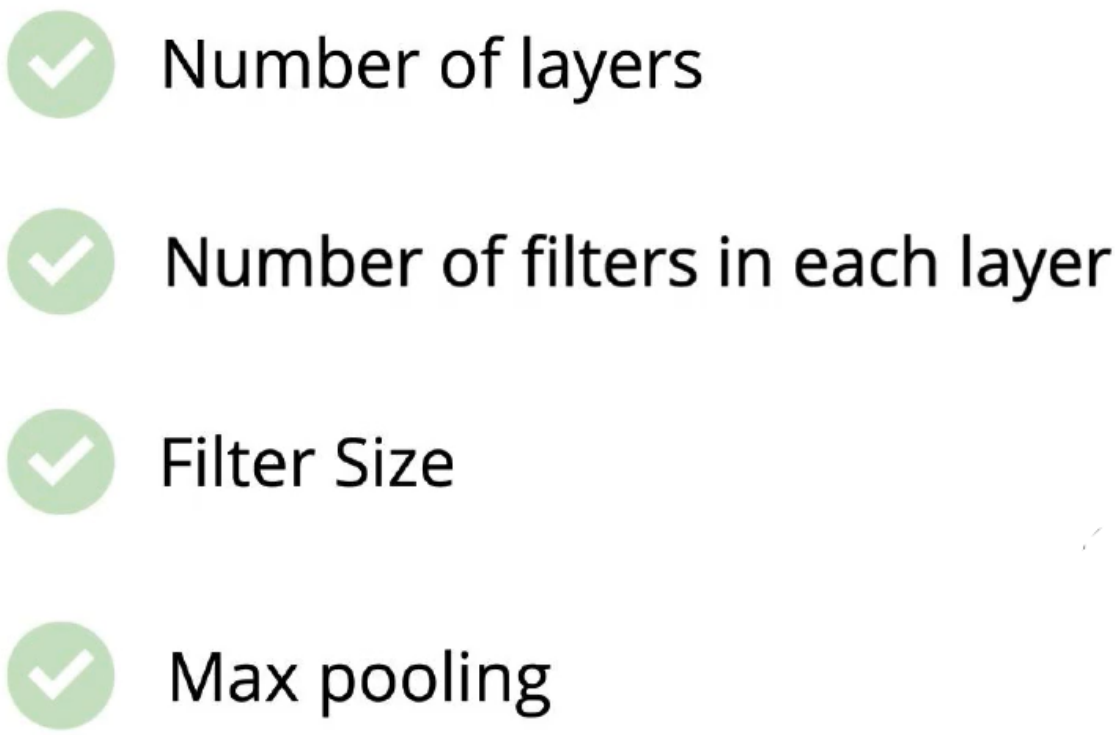
Classification Task: Here we predict the class to which the image belongs out of thousands of different categories/classes.

Classification + Localization: Here our job is not just to identify the class of the object in the input image but also to show where exactly the object is in the image(the

Object Detection task: Here the difference from the Classification task is that now we have multiple classes that exist simultaneously in the image and we need to classify and localize each of these objects.

Instance Segmentation: Here we are looking for the exact contour of the object, we don't return a bounding box where some noise might be there in addition to the actual object instead we return the exact contour of the object.

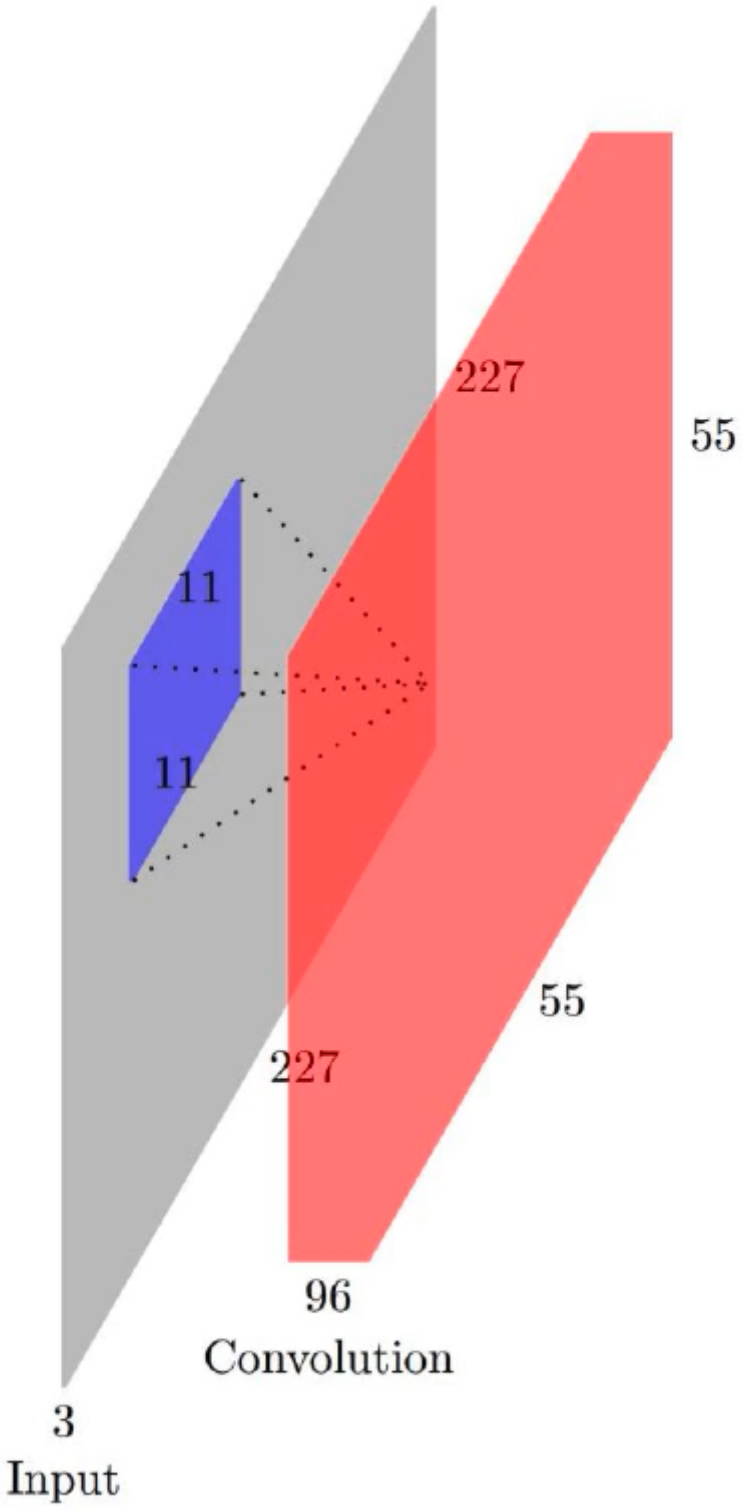
For all these tasks, we use CNNs and we need to make several decisions(for the CNN architecture to use) like the following:



The overall CNN architecture depends on the choices that we make for the above-mentioned points. We can't find the optimum values of all the hyper-parameters for all the tasks that we have. So, we prefer to use the standard tried and tested architectures. There are a few standard architectures that are available in all of the Deep Learning frameworks and in practice, we can just call a method and the architecture would be loaded into an object ready to use. One of the standard architectures is AlexNet which is discussed below.

ImageNet image classification competition) are of size **‘227 X 227 X 3’**(width X height X number of channels)

In the first layer, there are 96 filters each with the filter size **’11 X 11 X 3’**. Each such filter will give one 2D output, so we get **‘96’** 2D outputs, so the depth of the output is



Width is computed using the below formula:

$$W = \frac{W_1 + 2P - F}{} + 1$$

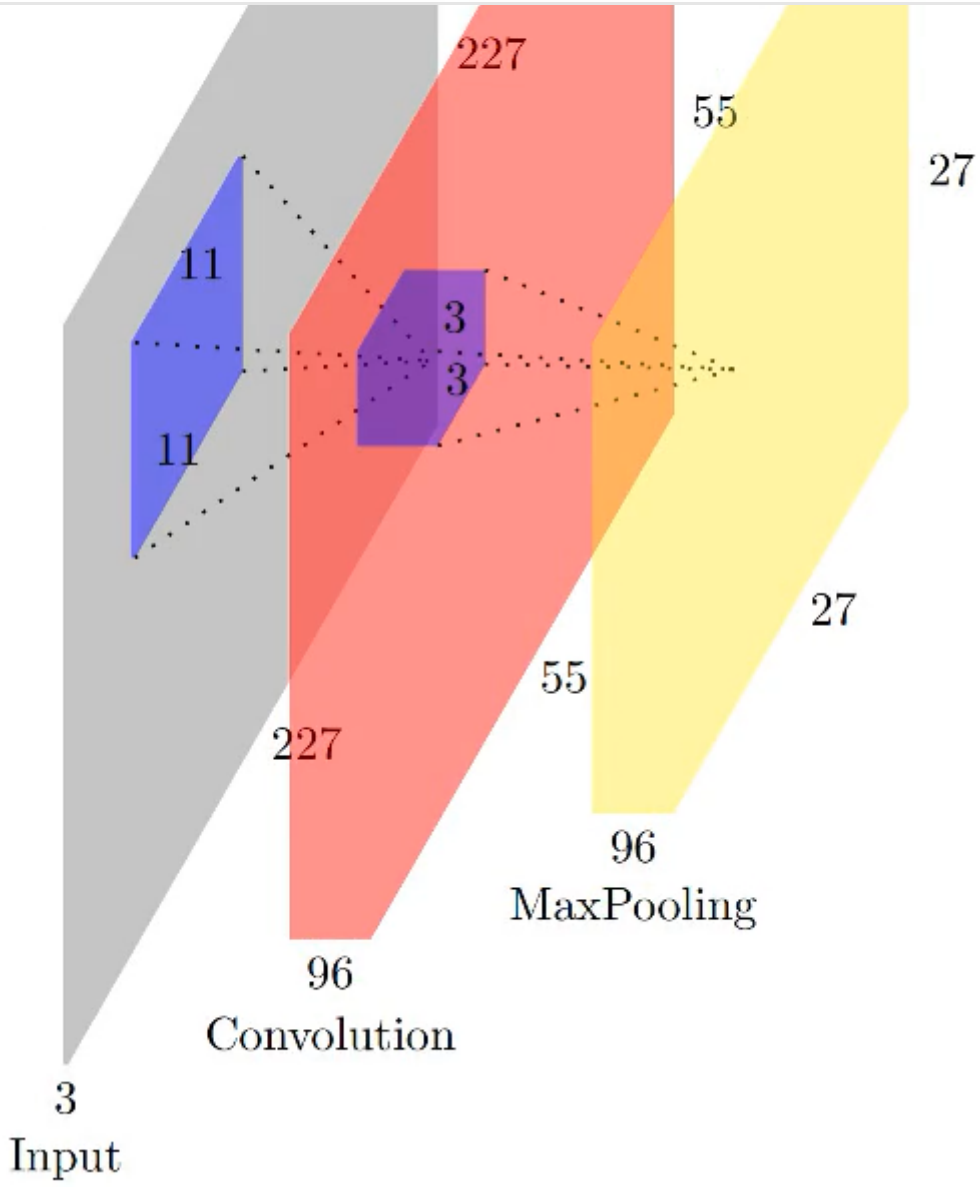
$$\text{Input: } 227 \times 227 \times 3$$
$$\text{Conv1: } K = 96, F = 11$$
$$S = 4, P = 0$$
$$\text{Output: } W_2 = 55,\ H_2 = 55$$

Input width is **227**, padding(P) is **0** and stride(S) is **4**, filter size(F) is **11**, using these values, we get the output width as **55**. And since the width and height are equal, to begin with, and we use symmetric filters, we get the height also as **55**. So, as the output of the first layer, we get the volume of size **'55 X 55 X 96'**.

Total Number of parameters in this layer: We have **96** filters, each filter is of size **'11 X 11 X 3'**, so the total number of parameters would be = **'96 X 11 X 11 X 3'** which comes out to be near to 34k.

Input dimensions are **'227 X 227 X 3'** and the dimensions of the first hidden layer is **'55 X 55 X 96'**. **If we have used fully connected neural network**, then the **total number of parameters** would have been = **'227 X 227 X 3 X 55 X 55 X 96'** which is of the order of **10^8 or 10^9** weights in the first layer whereas the same task is achieved using **34k** parameters using a CNN because of sparse connectivity and weight sharing.

After this first layer, we apply the ReLU non-linearity on the output of the previous layer(which acts as pre-activation) and then we have the max-pooling layer where filter size is of size **'3'**(that is we look at a **'3 X 3'** region and pick up the max value from that region), this **max-pooling operation would happen independently for all the channels**, that means our input has **96** channels, the output of max-pooling is also going to have **96** channels. Stride value used for this max-pooling layer is **2** because of which the output width and height would be roughly half of the width and height of the input for this layer.



Number of Parameters, in this case, would be **0** as in max pooling, we are not doing any weighted aggregation or multiplication of that sort, we are given some numbers and we are taking just max of those numbers.

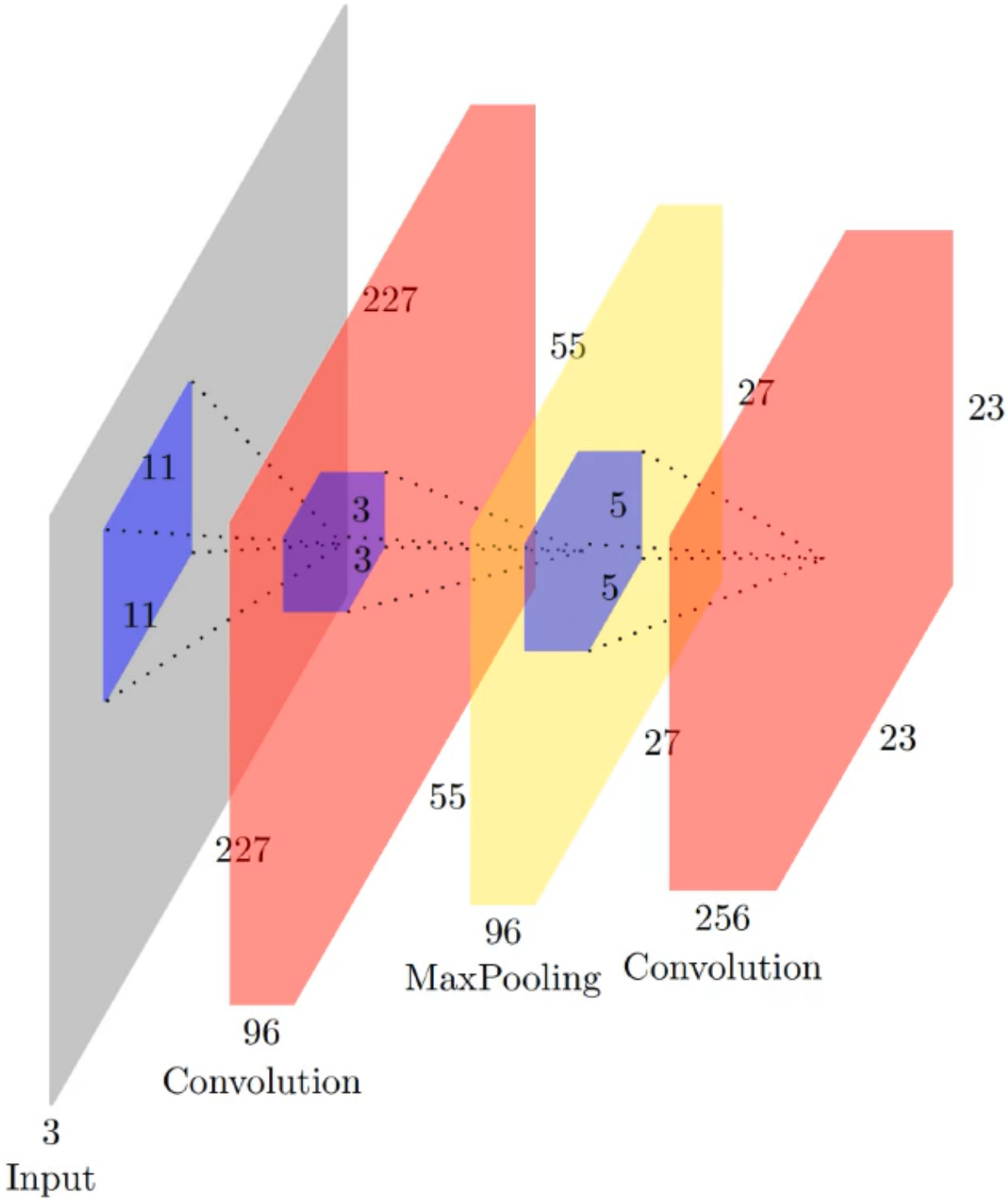
Max Pool Input: $55 \times 55 \times 96$

$F = 3, S = 2$

Output: $W_2 = 27,\ H_2 = 27$

Parameters: 0

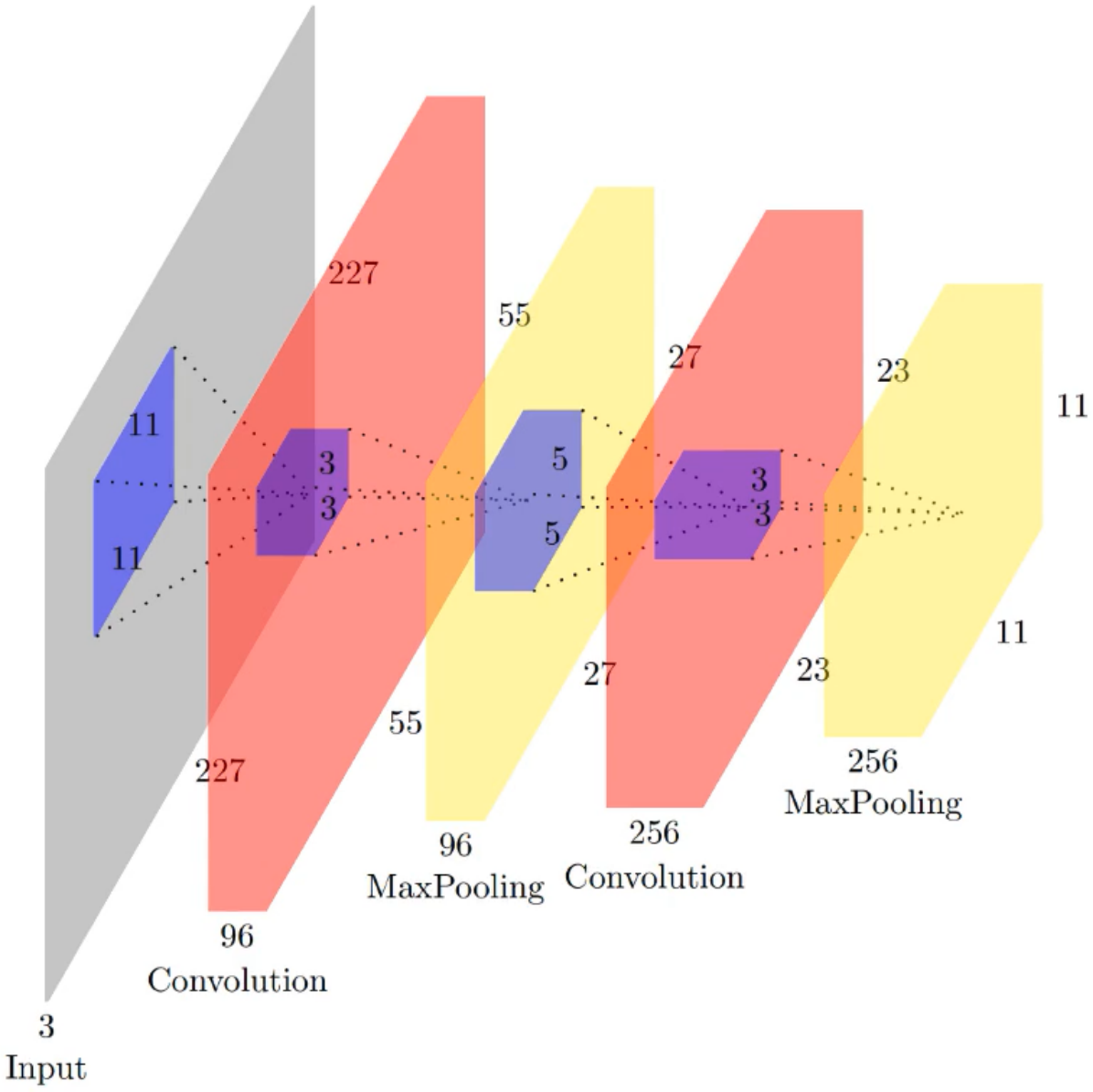
used in this layer are **256**(that means the depth of the output would also be 256), the filter size is **'5X 5'**, padding used is **0** and the value of stride is **1**. Using these values in the standard formula to compute the output width and height, we get the output volume of dimensions **'23 X 23 X 256'**.



Total number of parameters in this case would be: **'5 X 5 X 96 X 256'** = 0.6M

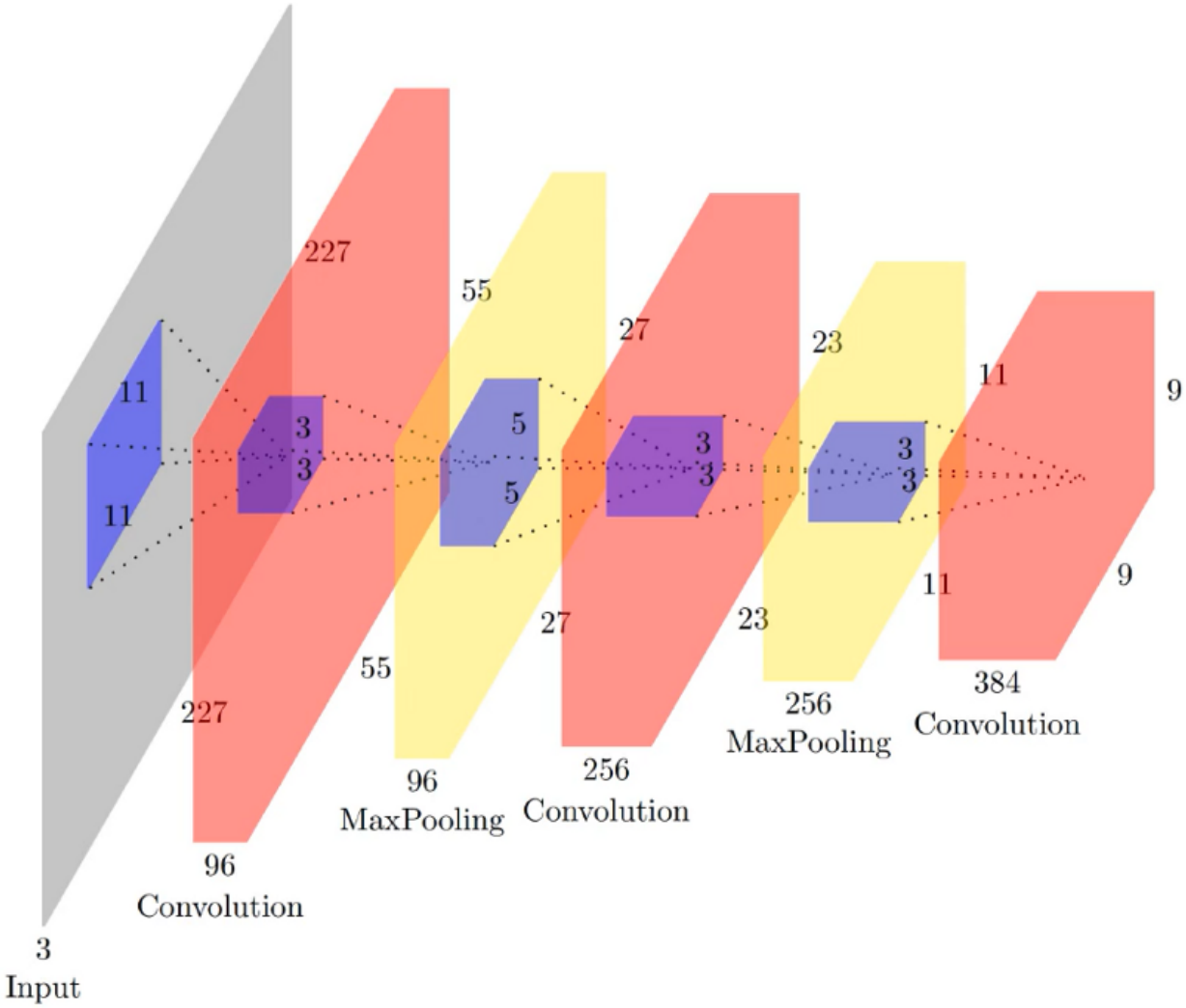
$$\text{Conv1: } K = 256, F = 5$$
$$S = 1, P = 0$$
$$\text{Output: } W_2 = 23,\ H_2 = 23$$
$$\text{Parameters: } (5 \times 5 \times 96) \times 256 = 0.6M$$

After this, we again use the max pool layer with a stride value of **2** which would just reduce the input dimensions to half, the number of channels/depth would remain the same as max-pooling is per channel operation and the number of parameters for this max-pooling layer is going to be **0**.



$$\text{Max Pool Input: } 23 \times 23 \times 256$$
$$F = 3, S = 2$$
$$\text{Output: } W_2 = 11,\ H_2 = 11$$
$$\text{Parameters: } 0$$

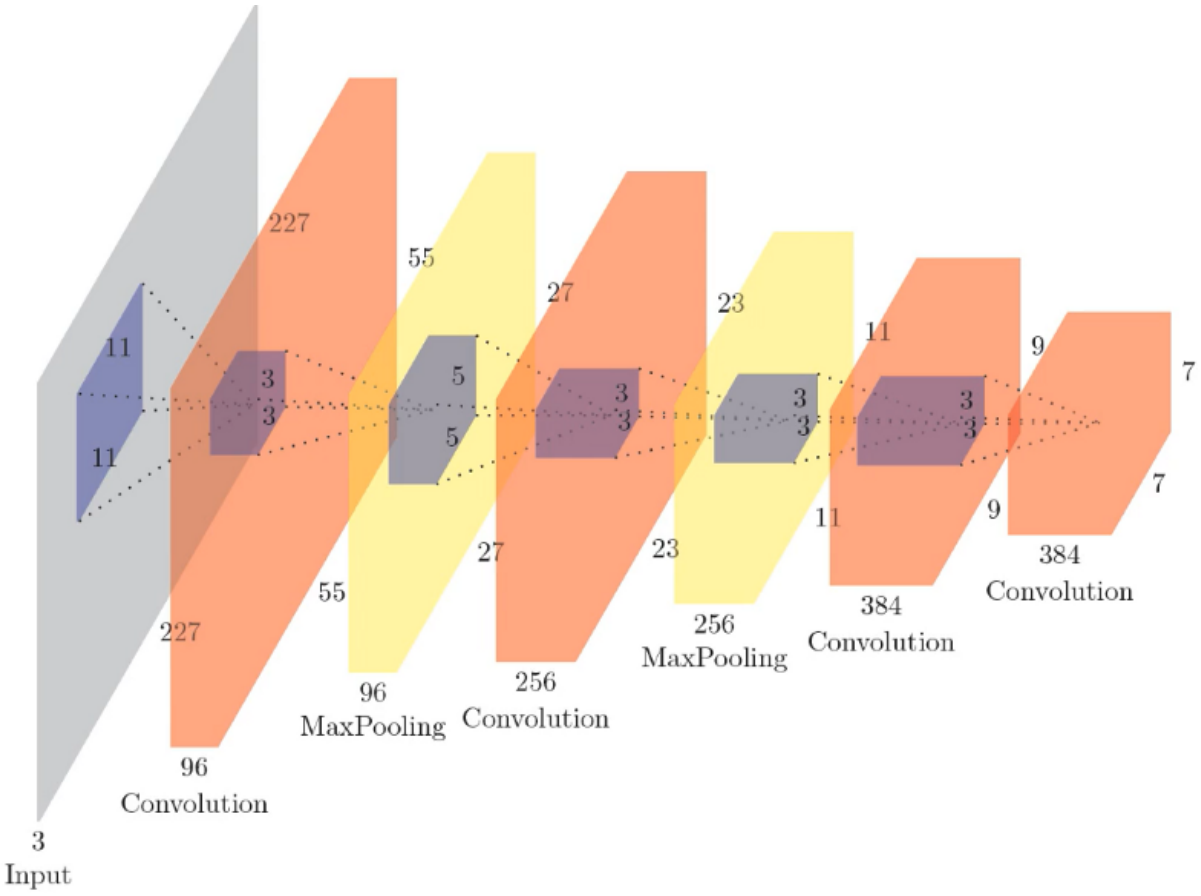
After this max-pooling layer, another convolutional layer is there where the filter size is **3**, the number of filters are **384**, so after applying the standard formula for computing the output width and height considering appropriate value for Padding and stride as in the below snippet, we get the output volume dimensions as **9**.



$$\text{Input: } 11 \times 11 \times 256$$

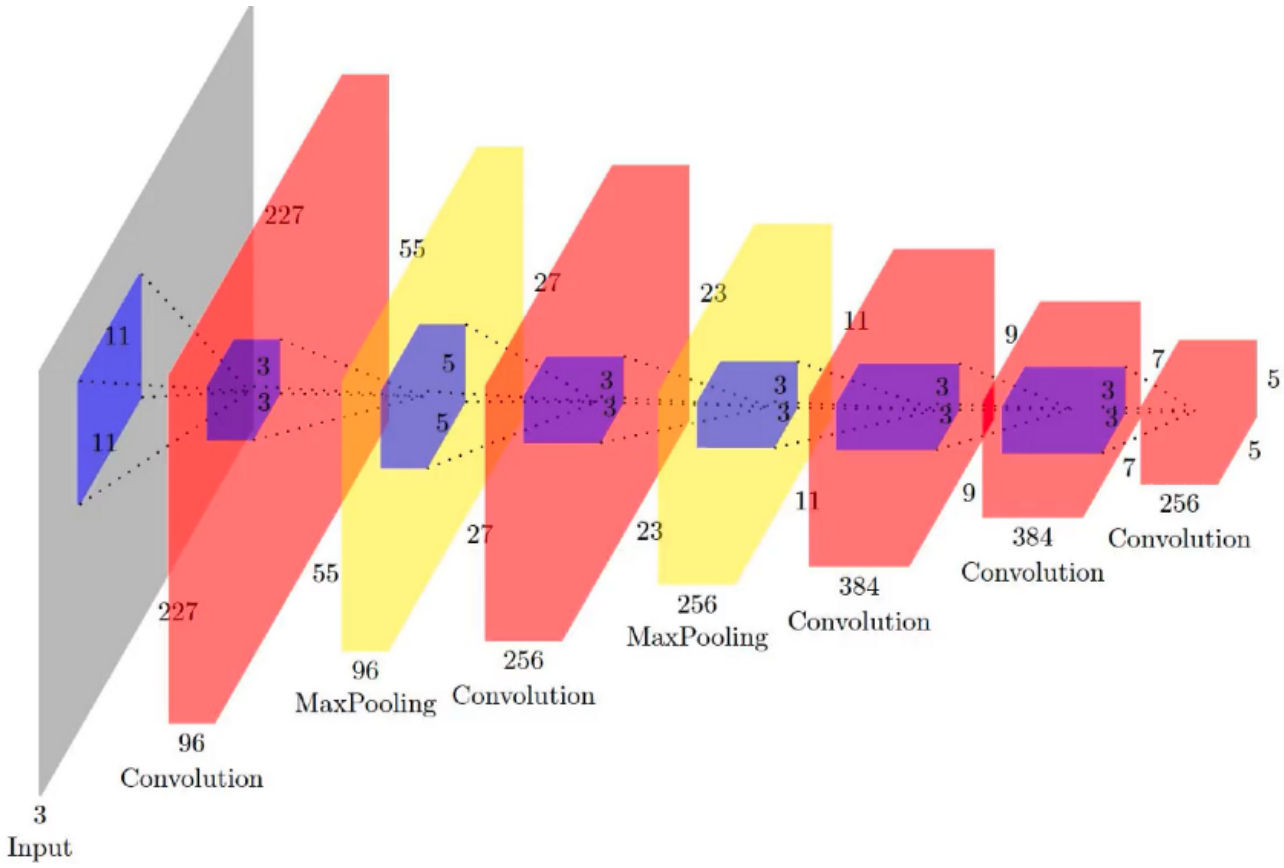
$$S = 1, P = 0$$
$$\text{Output: } W_2 = 9,\ H_2 = 9$$
$$\text{Parameters: } (3 \times 3 \times 256) \times 384 = 0.8M$$

After this, we have another convolutional layer:



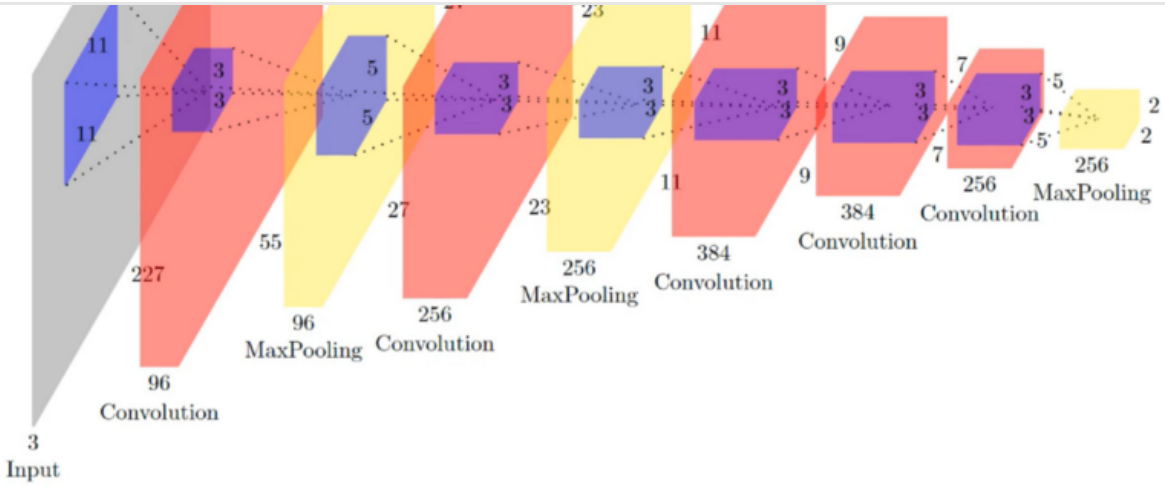
$$\text{Input: } 9 \times 9 \times 384$$
$$\text{Conv1: } K = 384, F = 3$$
$$S = 1, P = 0$$
$$\text{Output: } W_2 = 7,\ H_2 = 7$$
$$\text{Parameters: } (3 \times 3 \times 384) \times 384 = 1.327M$$

After this we have another convolutional layer, where we used **'3 X 3'** filters and use **256** such filters:



$$\begin{gathered}\text{Input: } 7 \times 7 \times 384 \\ \text{Conv1: } K = 256, F = 3 \\ S = 1, P = 0 \\ \text{Output: } W_2 = 5,\ H_2 = 5 \\ \text{Parameters: } (3 \times 3 \times 384) \times 256 = 0.8M\end{gathered}$$

After this, we have the max-pooling layer with a stride of **2** which reduces the input dimensions to half.



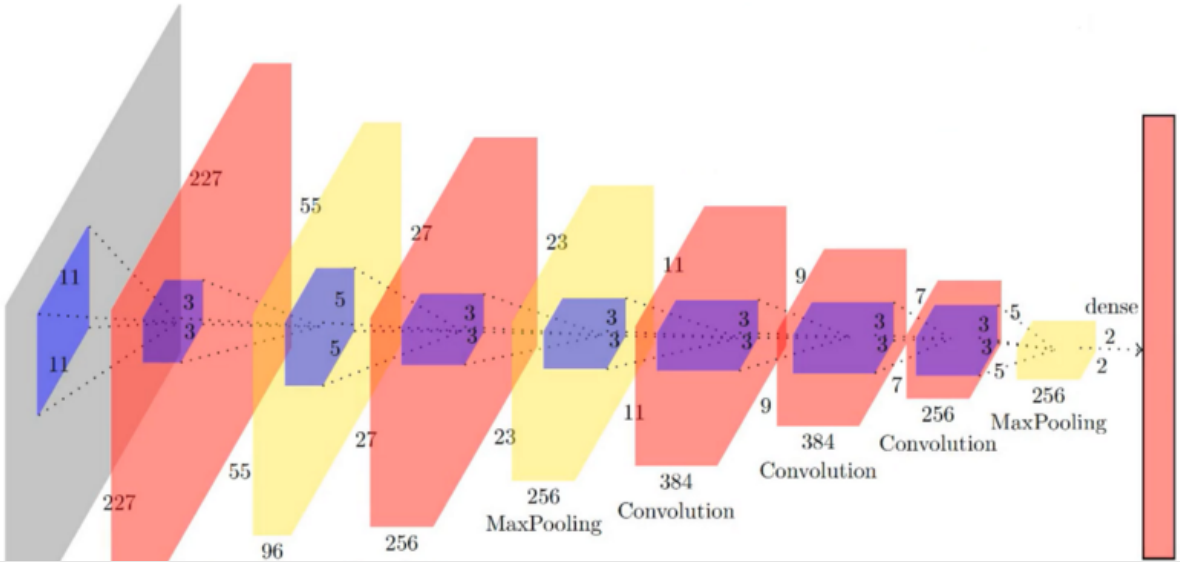
Max Pool Input: $5 \times 5 \times 256$

$F = 3, S = 2$

Output: $W_2 = 2,\ H_2 = 2$

Parameters: 0

We flatten this last layer of dimensions **'256 X 2 X 2'** into a single vector of size **'1024'** dimensional. This vector is then connected to another **'4096'** dimensional vector via a fully connected layer.

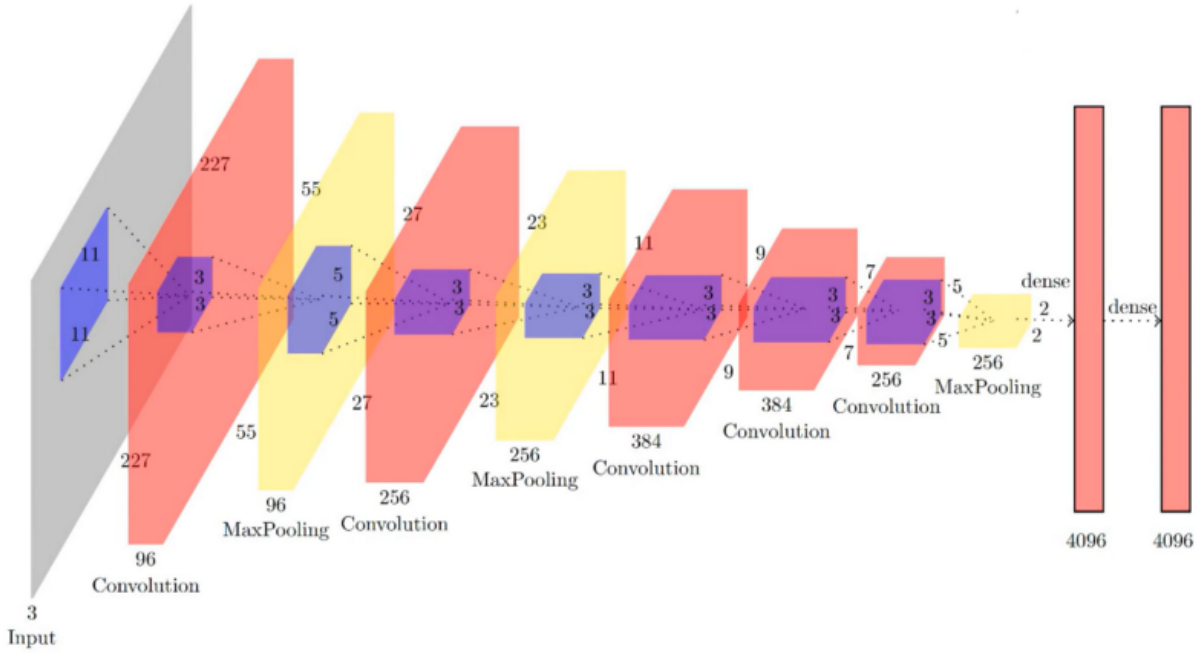


Input

FC1

Parameters: $(2 \times 2 \times 256) \times 4096 = 4M$

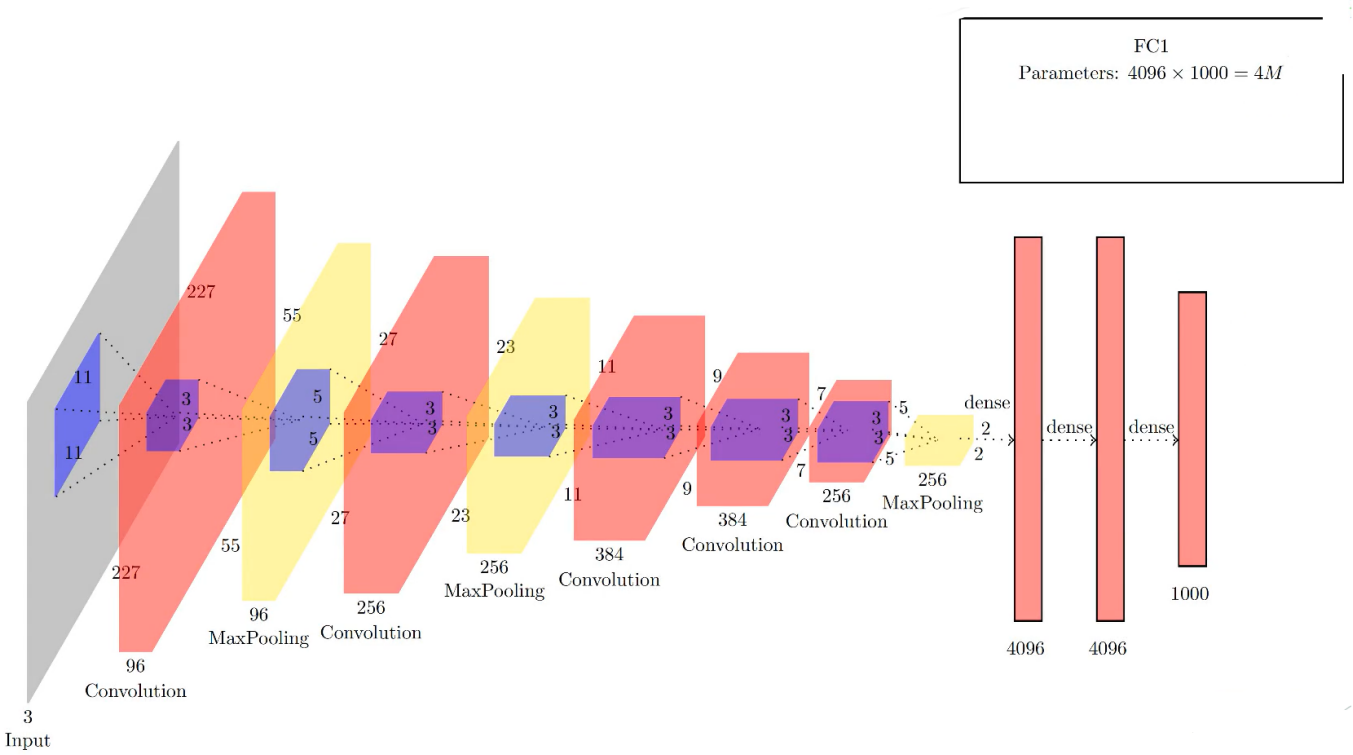
After this, we have another '**4096**' dimensional vector, which is going to be fully connected with this layer.



FC1

Parameters: $4096 \times 4096 = 16M$

After this we have the final classification layer which has 1000 neurons:



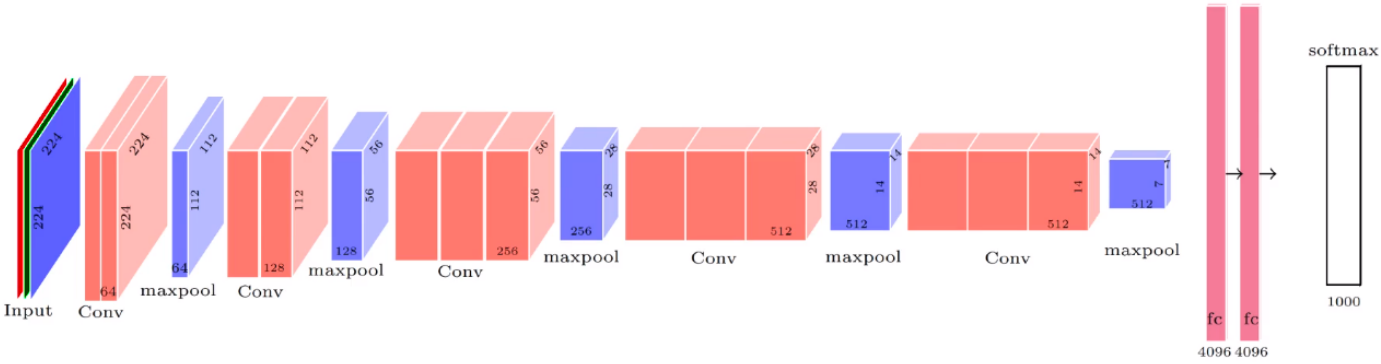
So, if we aggregate the parameters for all the layers, we get a total of 27.55M parameters.

Total number of layers in Alex Net = 8

We don't consider the Max-Pooling layers when counting the total number of layers as max-pooling layers do not have any parameters.

ZFNet also has 8 layers just like Alex Net, the only difference being that in some layers ZFNet has more filters and in some layers, the filter size is different.

architecture is depicted below:



VGG 16 Architecture

- Kernel size is 3×3 throughout
- Total parameters in non FC layers $= \sim 16M$
- Total Parameters in FC layers $= (512 \times 7 \times 7 \times 4096) + (4096 \times 4096) + (4096 \times 1024) = \sim 122M$

We pass the input image through a standard network and we get some representation of the image which we can then use with the fully connected layers to regress the coordinates of the bounding box.

Input (x) = image

Output (y) = a, b, w, h

Once we get the bounding box, we will crop that part of the image, use this as the input and now we need the text in this bounding box as the output.

We pass this cropped image(from the original image based on bounding box) through a standard architecture and we get some representation of the image which we can then pass through the RNN or Encoder-Decoder model which will return us the text.

Convolutional Network Convolution Neural Net Deep Learning Computer Vision

Artificial Neural Network